経営学研究論集
第39号　2013.9

ソフトウェア開発における研究開発と非研究開発
——ソフトウェアに係る会計基準のセントラル・ドグマの解体へ——

R & D and Non-R & D in Software Development
—— To deconstruct Central Dogma of Accounting Standards for Software ——

博士後期課程　経営学専攻　2011年度入学
長　田　芙悠子
OSADA Fuyuko

【論文要旨】

The purpose of this paper is to deconstruct central dogma of accounting standards, and to construct a new fundamental cognition. Therefore I propose a definite criterion for judgment to be capable of distinguishing between R&D and non-R&D in software development. For software product, it is capable of positioning particular softwares with setting a coordinate of life stage for function and technology. For software process, it clarifies morphologically and quantitatively how kind of difference at adaptive process of R&D and non-R&D in novelty of product. Through those studies, I realize a criterion for judgment based on novelty in both sides of product and process for software.

【キーワード】
・会計基準（Accounting Standards）
・ソフトウェア開発（Software Development）
・研究開発（Research and　Development）
・プロダクト＆プロセス（Product , Process）
・生産性（Productivity）

Ⅰ　はじめに

ソフトウェア開発は，研究開発ではない。正確を期するならば，ソフトウェア分野でも当然に研究開発はあるが，あくまで一部であり，大半のソフトウェア開発は研究開発ではない。これが，

論文受付日　2013年4月26日　　大学院研究論集委員会承認日　2013年6月3日

本稿の主張であり，論証することである。

ところが，不可解なことに，ソフトウェアに係る会計基準は真逆の認識[1]に立脚しており，ソフトウェア開発の大半を研究開発と看做している[2]。日本基準は特に顕著で，そもそも研究開発に係る会計基準の中にソフトウェアに係る基準を含めている（以下，「ソフトウェア会計基準」又は文脈で判然とする場合には単に「会計基準」「基準」とする）。しかし，それは日本基準に留まらず，アメリカ基準がそもそもの発端である[3]。

ソフトウェア並びにソフトウェア開発の実態を少しでも知っていれば判然としたことであるにも関わらず，会計関係者の認識は乖離が甚だしい。インタンジブルズ研究で著名なバルーク・レブ（Baruch Lev）さえ，ソフトウェア資産の認識において，開発原価の範囲問題ではアメリカ基準に異議を唱えているが[4]，ソフトウェア開発（の大半）が研究開発であるという誤認は共有し，疑義を呈していない[5]。また，日本におけるソフトウェア会計の代表的な研究者である櫻井通晴は，比較的最近の論稿でも「基準」3（2）に関して「逆に言えば，ソフトウェア開発にも研究開発に該当しない開発費があることを正式に認めた」[6]という見解を表明しているが，それ以上に本格的な議論を深めていない。なお，櫻井に関しては後ほど改めて取り上げる。

本稿の目的は，ソフトウェア会計基準の全面刷新に向けて，そのセントラル・ドグマ[7]を解体し，新たな基底的認識を構築することである。そのために，ソフトウェア開発における研究開発と，ソフトウェア開発における研究開発ではない一般的な開発（以下，非研究開発と概ね略記）を明確に区別可能とする判断規準を考案することである。

その意義は，これまでの混濁した財務情報によるミスリードを是正することである。現行のソフトウェア会計基準に基づいて処理された財務情報は，取り立てて新奇性のないソフトウェアを研究開発の成果と看做すことにより，ソフトウェアに関して「紛らわしい」財務情報となっている。他方で，新奇的な研究開発ではない活動を研究開発と看做すことにより，研究開発費が「実態以上に」計上された財務情報となっている。研究開発費は，ミクロ的には研究開発費売上高比

---

1 「認識」には会計特有の概念並びに用法があるが，本稿ではより広く一般的な意味で「認識」という用語を概ね用いていると了解されたい。但し，会計的な脈絡では会計の概念として使用している。

2 ソフトウェア会計基準では，*ソフトウェアの研究開発≒ソフトウェアの非研究開発 ⇒ ソフトウェア開発≒研究開発*，と看做している。それに対して筆者が論証したいのは，*ソフトウェアの研究開発≠ソフトウェアの非研究開発*，ということである。なお，本稿では，会計基準におけるソフトウェアに関する基本的な捉え方を問題にしているので，「市場販売目的」等の下位分類以前のソフトウェアを総体として取り上げる。

3 ソフトウェア開発の大半を研究開発と看做していることは，IAS38 無形資産も同類であり，大同小異である。

4 Lev, Baruch（2001）邦訳 P.106

5 Lev, Baruch（2001）邦訳 P.106，P.118

6 櫻井通晴（2012）P.112

7 セントラル・ドグマとは，体系的な教義において，基底的であり且つ全体を貫く教義のことである（典型例はカトリックの三位一体説であり，周知の通り，プロテスタントにとって教義的には最大の争点であった）。ソフトウェア会計基準においては，ソフトウェア開発の大半を誤って研究開発と看做す規定が，それに相当する。

率という経営指標により企業の成長性を判断する重要な情報となっており，マクロ的には研究開発費GDP比率という経済指標により一国経済の成長性を判断する重要な情報となっている。ところが，その研究開発費に研究開発ではない「模倣的」な活動の費用が混在しているとすれば，「嵩あげ」された数値により，判断はミスリードされ続けてきたことになる。これは速やかに是正されなければならない。本稿は，そのための明解な判断規準を提示するものである。それにより，ソフトウェア開発における研究開発と非研究開発を適正に識別した情報が企業活動の信頼し得る情報となり，その開示がステイクホルダーへの適切な情報開示となることを目的としている。更に敷衍すれば，正確な情報こそが経済への貢献（良導）になる，と考える。

本稿の構成概要は，次の通りである。まず第1に，予備的考察として，アメリカ基準における不当な判断規準の策定がなされた経緯とその内容をコンパクトに捉え直し，それを踏まえて現行のソフトウェア会計基準の問題点を挙げ，更に会計基準に対して呈示された数少ない疑義を吟味する。第2に，研究開発と非研究開発を区別するために，ソフトウェア・プロダクトの位相を捉える枠組みを考案する。まず，ソフトウェア・プロダクトのライフステージ座標を設定する。次に，それを適用することで，ソフトウェア・プロダクトのライフステージの測位が可能になることを明らかにする。第3に，同様の目的で，ソフトウェア・プロセスの様相を考察する。まず開発プロセスそのものが研究開発であるソフトウェア開発のプロセス・イノベーションを挙示する。次に，プロセスそのものは研究開発とは言えないが，プロダクトが新奇的か否かで，研究開発と非研究開発のプロセスが形態的に相違があることを明らかにする。更に，それがプロセスの定量的差異となって現れることをも明らかにする。こうして，プロダクトとプロセスの両面から総合的に，研究開発と非研究開発の区別を明らかにすることにより，筆者の主張の論拠を具体的に証示する。

Ⅱ　予備的考察

1　アメリカ基準における不当な判断規準の策定

アメリカの販売用の自社開発のソフトウェア会計基準等は，年代順に挙示すれば次の通りである[8]。

・1969年　内国歳入庁（IRS）　内国歳入庁歳入手続69-21（Rev. Proc. 69-21）

・1974年　財務会計基準審議会（FASB）　財務会計基準書第2号（SFAS No.2）

・1975年　財務会計基準審議会（FASB）　FASB解釈書第6号（FIN No.6）

---

[8] 中村恒彦（2003）P.4。なお，各原名はカッコ内の通りである。内国歳入庁（Internal Revenue Service），財務会計基準審議会（Financial Accounting Standards Board），証券取引委員会（Securities Exchange Committee），アメリカ公認会計士協会（American Institute of Certificated Public Accountants），歳入手続（Revenue Procedure），財務会計基準書（Statement of Financial Accounting Standards），解釈書（Financial Interpretation），適用指針（Technical Bulletin），財務報告通牒（Financial Reporting Release），問題提起書（Issue Paper），公開草案（Exposure Draft）。

・1979年　財務会計基準審議会（FASB）　FASB適用指針第79-2号（FTB No.79-2）

・1983年　証券取引委員会（SEC）　財務報告通牒第12号（FRR No.12）

・1984年　アメリカ公認会計士協会（AICPA）　問題提起書

・1984年　財務会計基準審議会（FASB）　公開草案

・1985年　財務会計基準審議会（FASB）　財務会計基準書第86号（SFAS No.86）

最初は，「ソフトウェア開発費は，多くの面で1954年制定の内国歳入法第174条に該当する試験研究費と非常によく似ている」[9]という歳入手続69-21の規定から始まっている。しかし，どのように「多くの面で」「非常によく似ている」のか，説明はなされていない。その取り立てて根拠のない類推適用が，ソフトウェア会計基準の今日に到るも解消されていないセントラル・ドグマの発端である。続くSFAS No.2では，「コンピューター・ソフトウェアは，多種多様な目的のために開発される。それゆえ，個々のケースについては，ソフトウェアが開発される活動の性質を第8-10節の指針に照らして判断し，当該ソフトウェア原価がそれに該当するか否かを決定すべきである。たとえば，販売のために新奇で高度なコンピューター・ソフトウェアの能力を開発する活動は，本基準書でいうところの研究開発に該当する」[10]（傍点：引用者），と規定している。第8-10節の指針は新奇性に関する指針であり，この限りでは明解な規定のはずであった。新奇性を有するものは研究開発であり，新奇性を有しないものは研究開発ではない，というようにである。「ただし，SAFS No.2は，ソフトウェア原価の中には研究開発に該当する活動と該当しない活動があるとも指摘」[11]することにより，再び錯綜・混迷の道に入り込んでしまったと言える。

それ以降は，周知の通り，新奇性の指針は後退し，「研究開発に該当する」か否かをソフトウェア開発の工程で区分することになり，その根拠として「技術的実現可能性（technological feasibility）[12]」が枢要の位置を占めるようになる[13]。しかしこれは，問題の誤った解法である。研究開発の判断規準は，あくまでも製品又は製法の新奇性である。このことは，ソフトウェアであろうとも何ら変わらない。製品又は製法に新奇性のないソフトウェア開発は，研究開発ではないのである。ところが，ソフトウェアだけは何故か，新奇性のない，非研究開発にも「研究開発に該当する」部分（要素）があるという無理な捉え方がなされた。そのため，開発工程や「技術的実現

9　同上 P.5

10　同上 P.5-6

11　同上 P.6

12　SFAS No.86 Research and Development Cost of computer Software 4. における「技術的実現可能性」の確定の証拠は a. 又は b. である。a. 詳細プログラム設計を含む製作の場合，(1) 製品制作のための技術的な利用可能性，(2) 詳細プログラム設計の完了及び詳細プログラム設計と製品設計との首尾一貫性，(3) 詳細プログラム設計の開発上の不確実性の解消。b. 詳細プログラム設計を含まない製作の場合，(1) 製品設計及び作業モデルの完了，(2) 作業モデルの完了及び作業モデルと製品設計との首尾一貫性，である。

13　問題提起書並びに公開草案では販売可能性と財務可能性も挙げられていたが，SFAS No.86では削除された。なお，歴史的な考察としては丁寧なトレースが必要であるが，本稿は問題の剔抉を主眼としているので，簡略に留める。同上 P.10-45 以外にも，西澤脩（1991），櫻井（1993）等参照。

可能性」といった「判断規準」が連鎖的に導出されたのである。しかしこれも，非適合的な論理展開である。言うまでもなく，研究開発では技術的実現可能性は重要な関心事であるが，それはあくまで研究開発の成否の判断規準である[14]。技術的実現可能性が確保されたからといって，研究開発が研究開発でなくなるわけではない。当該研究開発が完了するまで，研究開発である。そうであるにも関わらず，ソフトウェア開発だけ何故，技術的実現可能性の確保というファクターによって，それまでの研究開発が研究開発でなくなるのか。成否によって基本的な特性が変わることはあり得ない。ソフトウェアに関しても，研究開発であれば，技術的実現可能性が確保される前も，確保された後も，何ら変わらず研究開発である。研究開発でなければ，始めから終わりまで研究開発ではない。何れの場合も，技術的実現可能性は開発の成否の判断規準ではあるが，研究開発か否かという基本的な特性の判断規準ではない。開発工程による区分はその派生系であるが，研究開発であれば何れの工程で技術的実現可能性が確保されるかは一律ではないし，研究開発でなければ余程杜撰な開発計画でない限り，予め技術的実現可能性は確保されているのである[15]。そもそもプロダクト（製品）を不問に付して，工程の区分というプロセス（製法）だけを決定的な規準とするだけでも片面的ではないか[16]。こうした問題のコンタミネーション（contamination）と，資産あるいは費用の何れとすべきかという会計処理の議論が混然となって錯綜し，出来上がったのがSFAS No.86である[17]。

2　日本の現行のソフトウェア会計基準の問題点

日本のソフトウェア会計基準は，販売目的のソフトウェアに関して，一言で言えば，アメリカ基準を模倣し，更に誤謬を増幅させたと言える[18]。模倣した点は，ソフトウェア開発を研究開発と不当に拡大解釈し，研究開発であるものはもとより，非研究開発にも研究開発的要素があるということ，その有無を判断するのに「技術的実現可能性」が確保されたか否かを規準とし，開発工程で区分するということである。誤謬を増幅した点は，開発工程における「技術的実現可能性」の判断時点をアメリカ基準より下流に引き下げ，研究開発を「最初に製品化された製品マスター

14　成否の判断規準として，重要であるが，あくまで規準の1つである。技術的には実現可能でも，投資効果等で成功の見込みなしと見做し，中止あるいは製品化の断念等はあるからである。

15　一般的なソフトウェア開発でもプロジェクトの失敗は少なくないが，それは別の種々の要因による（別稿「ソフトウェア開発の失敗に関する会計処理案―ソフトウェアの仕損を会計ではどのように捕捉すればよいか―」『明治大学　専門職大学院研究論集』第5号で既に取り上げており，それを参照）。なお，研究開発でない一般的なソフトウェア開発においても，技術的実現可能性に疑念がある場合は，開発の早期にフィージビリティ・スタディ（必要ならば実装を含めて）を行うものである。

16　SFAS No.86の「技術的実現可能性」の要件には，一部プロダクトに関する規定も含まれてはいるが，形式的な要件に過ぎないので，取り扱っていないに等しい。

17　IBM社の影響力によるバイアスといった刻印が顕著だが，政治力学の問題なので割愛する。

18　櫻井通晴（1999）は，「移植」（P.4）という言い方をしているが，カスタマイズもしており，且つそれが改悪の方向になっている。

の完成」までとしたこと，研究開発の判断規準に「販売の意思」をも含めたことである。これによって，研究開発ではない範囲はごく僅かに狭められた。

西澤脩は，会計基準をほぼ忠実に祖述している。「同じソフトウェアでも，研究開発目的のソフトウェア（専ら当該研究開発プロジェクトに使用され，他の研究開発プロジェクトに転用できないもの）は，ライフサイクルの全過程が研究開発とされる。問題は，非研究開発目的のソフトウェアである。この種のソフトウェアについては，開発過程までが研究開発とされ，生産以後は研究開発の対象外となる」[19]（傍点：引用者）。市場販売目的のソフトウェアに関しては，「最初に製品化された製品マスター」の完成「時点までの制作活動は研究開発と考えられるため，ここまでに発生した費用は研究開発費として処理する」[20]（傍点：引用者），としている。市場販売目的のソフトウェアに限らず，自社利用のソフトウェアに関しても誤って同様の記述をしている[21]。総じて西澤は，非研究開発目的のソフトウェアであるにも関わらず，「開発過程までが研究開発とされ」る，あるいは製品マスター完成時点までの「制作活動は研究開発と考えられる」などと，自明の如く断定するだけで，理由等の説明を何ら行っていない。何故，研究開発目的ではないにも関わらず，ソフトウェア開発のある範囲が「研究開発」と看做されるのか。一般的なソフトウェア開発は，開発工程によって「研究開発活動」と「商業生産活動」の区分などがあるわけもなく，全てが「商業生産活動」である。受託開発はそれに基づいて成り立っている。市場販売目的の開発でも同様である。研究開発を目的としていないのに，「実態」としてそうだと看做すのであれば，それ相当の根拠ないし理由付けがあって然るべきではないのか。しかし，何も説明せず，会計基準をただ鵜呑みにしているだけである。それは西澤に限ったことではなく，管見の限り，ほとんど自明の理であるかの如くに，不問に付せられたまま，それを当然の前提として資産認識される原価の範囲問題等が論じられている。このように，アメリカ基準の誤謬を模倣したことが，第１の問題である。

第２に，「著しい改良」に関する会計処理の非合理性は，上記の研究開発と看做す誤りの派生系である。「著しくない改良」が資産の増加となるのに，それよりも更に資産性の増す「著しい改良」が大半は資産計上されず，費用処理されるという，著しく合理性を欠く規定とならざるを得なかったのは，「著しい改良」を研究開発と誤って看做した結果である。「著しい改良」に関する実務指針の例示に関しては，何れも研究開発でないことが明らかである。精細にケース分けす

---

19 西澤脩（2003）P.31。なお，同ページの図表 2-1「ソフトウェアのライフサイクルと研究開発の範囲」によると，ソフトウェアのライフサイクルを大きく「制作過程」と「付随過程」に区分し，次に「制作過程」を「計画」，「設計」，「開発」，「生産」に区分しているので，「生産以後」とは，「生産」（マスター制作とマスター複製）と「付随過程」（「アフターサービス」（顧客支援，保全，機能強化）と「廃棄」（除去・評価）を意味する。

20 同書 P.41

21 同書 P.38。そして，市場販売目的と自社利用の会計処理について費用と資産の範囲を同じように図示しているが，それは実務指針を誤って解釈したものである。正しくは自社利用の場合には資産要件を充足していれば，開発の全範囲を資産計上とするのである。

れば，採用する機能や技術のライフステージ如何によっては，研究開発であり得るものもあるが，その立ち入った考察は後ほど行う。

第３に，市場販売目的と自社利用の会計処理に大きな相違があることである。市場販売目的のソフトウェア開発の大半をリスクの高い研究開発と看做し，他方では自社利用のソフトウェア開発にはリスクを想定しておらず，それほど研究開発的要素が多くあると看做していないため，資産認識の範囲が著しく異なっている。しかも資産認識の判断規準として，市場販売目的では「技術的実現可能性」としているのに対して，自社利用では「将来の収益獲得又は費用削減の確実性」とし，「開発」と「利用」という全く異なる側面に照準を当てた規準となっている。そもそもの予断として，市場販売目的のソフトウェアは「新奇性」があり且つ技術的に高度な開発であるという「思い込み」が伏在しているが，実態は必ずしもそうとは言えない。利用以前に，開発においても自社利用ソフトウェアの方が，新奇性があり技術的にも高度なものは少なくない[22]。そしてまた，開発と利用とは全く別次元であるが，自社利用のソフトウェアの利用による効果の「確実性」が利用におけるリスクの想定だとすれば，事前に将来のリスクを「確実に」回避できることを要件とするということは，リスクがないと看做しているに等しい。しかし，利用におけるリスクもまた存在するのである。

誤謬を増幅した点の１つとして，「技術的実現可能性」の判断時点を後方化させたのは，製造業一般における研究開発並びに研究開発後の製造と無理に対応付けようと意図したからである。ソフトウェア開発には言うまでもなく研究開発と，研究開発過程のない非研究開発がある。研究開発である場合には製造業一般と対応付けても一応間違いではないが，研究開発でない場合にはそもそも対応付けは成り立たない。製造業においても研究開発がなく，全てが製造である業態はあり，強いて言えば，その製造に相当する。但し，定型性の該当範囲が異なり，そうであるが故にソフトウェアでは「製造」と言わず，開発と言うのである[23]。しかも研究開発である場合，研究開発後の製造は製造業一般の製造と比べれば，複製の容易性等からごく軽微な工程でしかない

---

22 別の例証を挙げる。Jones, Capers（1995）において，ジョーンズ自らが所属するSPR（Software Productivity Research）社の調査結果として，６つのソフトウェア分野に関するソフトウェアの成功と失敗を，６つの評価基準によりランク付けしている。ソフトウェア分野は，①システムソフトウェア（OS等），②軍需ソフトウェア，③情報システムソフトウェア（企業の業務システム），④外部委託ソフトウェア（③と同様のソフトウェアをソフトウェア企業が受託開発したもの），⑤市販ソフトウェア，⑥ユーザ開発ソフトウェア（例えば表計算ソフトにマクロを組み込んだシステムのようなもの），である。評価基準は，①スケジュール，②コスト，③品質，④利用性，⑤支援，⑥保守，である。この調査結果に拠ると，日本及びアメリカの会計基準における市場販売目的のソフトウェアに相当する①システムソフトウェアと，⑤市販ソフトウェアの成功が，総合的に他の分野に比べて遥かに優れた結果となっている。これは，本稿の論旨を支持する内容であり，会計基準において自社利用に比して，市場販売目的に関する技術的実現可能性が困難で，リスクが多く，研究開発的性格が濃厚と看做していることが実態にそぐわないことの一つの例証と言える。

23 ソフトウェア開発の中の１つの工程として，プログラム製造があるが，製造業一般における製造（全般）とは意味並びに対象範囲が異なる。

ことは無形のソフトウェアの特性によるもので，無理に同等ないし近似的な対応付けをする必要などないのである。

2つに，研究開発の判断規準として，技術的実現可能性に留まらず，市場におけるソフトウェア製品の「販売の意思が明らかにされること」までをも含めている。この点がアメリカ基準との大きな違いであるが[24]，開発におけるリスクと販売におけるリスクとを混同している。それ以外にも問題は少なからずあるが，本稿の主題との兼ね合いで，以上に留める。

3 研究開発類推適用への数少ない疑義の呈示

ソフトウェア会計基準制定前後に相当数の関連論文等が発表されているが，ソフトウェア開発を研究開発と看做すことに疑義を呈している先行研究は思いのほか少ない。

その数少ない研究の1つが，会計基準制定前に発表された，天明茂（1996）である。「汎用ソフトウェア」（会計基準の分類では「市場販売目的」のソフトウェアに相当する）に関して，次のように述べている。「よほど新規技術を伴う研究開発要素の高いものは別として，通常のソフトウェア開発においては基本設計以降に残された不確実性は，開発コストの増加，開発期間の延長，効率低下などソフトウェアの経済性に影響しても，ソフトウェアの技術的な成否に直接結びつかないのがほとんど（15）と言われる。こうした状況に鑑み，自社のソフトウェアの特性から，技術的実施可能性が確立される工程がほぼ決まっているような企業にあっては，当該特定工程以降の原価を資産計上の対象とすることも実務上の簡便法として認められよう」[25]。更に続いて，次のように述べている。「なお，資産計上の対象とされない費用についてFASが期間費用としていることには異論はないが，そのすべてを研究開発費とすることには疑問である。私見では研究開発要素の強いものは除いて，通常のソフトウェア開発に関わるものは売上原価として当該期間の売上高に対応させるべきものと考える。したがって製品マスターの開発費は研究要素の強い商品化決定前の研究開発費，通常のソフトウェア開発費のうち商品化決定前の期間原価，そして同じく商品化決定後の資産計上すべき原価に3分類されることになる」[26]。これらにおいて，「よほど新規技術を伴う研究開発要素の高いものは別として」，あるいは「研究開発要素の強いものは除いて」，「通常のソフトウェア開発」は研究開発と看做すべきではない，という見解が述べられている。更に精緻化されれば有力な見解たり得たであろうが，「研究開発要素」の詳細な内容

---

24 SFAS No.86では資産要件から除外されたが，公開草案では「市場可能性」と「経営者の支持」も要件とされていた。このうち「経営者の支持」が，日本基準の要件の「販売の意思」に相当するが，日本基準はごく限定して不完全な要件のみを継承した，という意味において誤謬を増幅させている。

25 天明茂（1996）P.92。なお，同稿は汎用ソフトウェアの資産性と共に，（長期）棚卸資産としての計上を妥当とする主張を行っているが，今日的には科目としては無形資産計上が通説となっており，筆者も同意見であり，棚卸資産論議には最早歴史的「意義」しかなく，棚卸資産であるコピー製造した製品と，マスターは別個のものであるので，立ち入らない。また，文中の注（15）は田宮治雄（1989）を参照している注記への指示である。

26 同上 P.92

に踏み込んだ進展はそれ以降にもみられなかった。なお，工程での区分という点ではアメリカ基準に囚われている，と言える。

もう1つは，会計基準制定後に発表された，櫻井通晴（1999）である[27]。「グローバル・スタンダードに一歩近づけた」[28]こと，また「統一的な判断基準を与えたものとして」「賛意を表する」[29]としながらも，基本的な構成等に関する疑問を呈している。まず，そもそも「ソフトウェアの会計基準を“研究開発費等”とする名称のなかで取り扱ったことが妥当であったかについては疑問が残る」[30]とし，続く「私見」でも次のように述べている。「第一に，「基準」では，研究開発費等に係る基準とされている。しかし，その実質的な内容がソフトウェアの会計基準であることは誰の目にも明らかである。つまり，本文から明らかなように，研究開発費に関する基準としては，“研究開発費は費用処理”と規定したにすぎなく，実質的にはソフトウェア会計基準である。であるとすれば，なぜ正直にソフトウェア会計基準としなかったか。明確な弁明が必要であったように思われる」[31]。しかし，制定関係者からの「弁明」は，管見の限り全くなされていない。

「第二に」挙げている理由は，「ソフトウェアを研究開発費の枠組みのなかでとりあげるのはあまりに時代に遅れているように思われる」とし，それはアメリカで「一九八五年の会計基準（FASB 第八六号）が研究開発費とソフトウェア開発費とを切り離して論じていることからもあきらかである」[32]としている。

---

27 櫻井は，同時期に同主題の論文を他にも発表しているが，基準を最も明確に批判しているのは，当論文である。

28 櫻井通晴（1999）P.4。なお，「グローバル・スタンダードはアメリカのそれであり」と断り書きをし，「日本企業とは企業環境が異なるところへの会計基準の移植は，ソフトウェア・ディベロッパーやベンチャー・ビジネスに対して若干の問題を提起した」（P.4）と続けている。

29 同上 P.6

30 同上 P.6

31 同上 P.6。なお，後続の箇所でも，「研究開発費の基準ではなく，「ソフトウェア会計基準」と題した基準を制定してほしかった」（P.6），と繰り返している。

32 同上 P.6。但し，櫻井が根拠とする歴史認識は支持できない。「ソフトウェアを研究開発費のなかで公式にとりあげたのは，アメリカではIRS（内国歳入庁）が一九六九年，FASB（財務会計基準審議会）は一九七四年のことである。当時はまだコンピュータ・ソフトウェアに関する知識も浅くソフトウェアの開発が困難を極めていたから，研究開発費会計の枠組みのなかでとりあげられたのは当然であった」とし（傍点：引用者），「しかし，一九八〇年代になると，ソフトウェアの開発は研究開発というよりは一種の生産活動であると解されるようになってきた」（P.6）という極端な年代的対比をしているが，その間に劇的な変化があったわけではない。従って，ソフトウェア開発という事象自体は，一般的な開発であれば，1969年ないし1974年においても研究開発ではなかった。また，「ソフトウェア資産性をめぐる問題は以前ほどの重要性を持たなくなっている」と述べ，その理由として「情報技術の環境は，旧来のレガシー・システムから現在ではクライアント／サーバ・システム（Client/Server System;C/S システム）に移行し」，それは「従来の汎用コンピュータによるレガシー・システムのように大幅なソフトウェア開発費は必要でなくなった」（P.4）としているが，明白に誤りである。C／S系は当初ローカルな比較的小規模なシステムに採用されたから，開発費が少なかっただけで，次第に基幹系システムにも普及拡大し，開発が大規模化するにつれて，開発費も増大したのである。些かも資産性が「重要性を持たなくなっ」たわけではない。

「第三に」挙げている理由は，次の通りである。「日本企業でソフトウェアを研究開発費のなかで取り上げることは，日本企業のソフトウェア開発の実態からすると，OSを開発している特定のコンピュータ・メーカーを除けば，現実の感覚と遊離しているといわざるをえない。現在のソフトウェア会計基準の制定に当たっては，少なくとも三種類のソフトウェア・ディベロッパーのソフトウェア開発環境を考慮すべきであった」[33]。「コンピュータ・メーカーが開発するOSはたしかに研究開発的要素が大である。しかし，ユーザー企業がソフトハウスに開発を依頼するのは，大方が研究開発的要素の少ない委託開発である。ソフトハウスの多くは研究開発的要素が少ないアプリケーション・ソフトウェアを受託している（後略）」[34]。「このように見ると，「意見書」はコンピュータ・メーカーの開発するOSを想定して制定されているようにすら思われる。その反面，日本における大多数のソフトウェアを開発しているソフトハウスの実態や意見は「意見書」にほとんど反映されていない」[35]，と断じている。業界事情に踏み込んで，傾聴に値する見解を述べているが，肝心なところでは支持できない。1つは，企業（ディベロッパー）単位の捉え方の粒度が粗いことである。コンピュータ・メーカーも，研究開発だけではなく，受託開発等もしている。また，「研究開発的要素が少ないアプリケーション・ソフトウェア」というが，アプリケーション・ソフトウェアでも研究開発はある。従って，あくまで個々のソフトウェア単位に，研究開発か非研究開発かを捉えなければならない。もう1つは，「研究開発的要素が少ないアプリケーション・ソフトウェア」というように，「少ない」ながらも「研究開発的要素」が含まれていると述べていることである。その「研究開発的要素」の多少とは具体的にどのような事態であり，どのように分析あるいは測定し得るのかを明確にし，「研究開発的要素が少ない」のではなく，非研究開発には研究開発的要素が含まれていないことを突き詰めなければ，会計基準の基本的な誤謬を打破し得ないのである。それ以降の研究を含めて，櫻井はそこまで進展させていない。

Ⅲ　ソフトウェア・プロダクトの位相

1　ソフトウェア・プロダクトのライフステージ座標

研究開発は，新奇的な製品又は製法を追求する活動である。ソフトウェアに関しても，何ら変わらない。そこでまず，ソフトウェア・プロダクト（製品）の新奇性をどのように捉えるか，を明らかにする。それによって，個々のソフトウェア開発が研究開発であるか，それ以外の一般的なソフトウェア開発であるかを明確に判断することができるからである。天明茂並びに櫻井通晴が，疑義を呈しながらも，具体化に到っていないことへのアプローチを試みたいのである。

筆者は，ソフトウェア・プロダクトの新奇性を捉えるために，ジェフリー・ムーア（Geoffrey Moore）が提唱するイノベーションの「ライフサイクル」論を援用する。それは，「新たなテク

33　同上 P.6

34　同上 P.6

35　同上 P.6

ノロジーに基づく製品が市場に受け入れられていくプロセスを，製品ライフサイクルの進行にともなって顧客層がどのように変遷するかという観点からとらえたもの」[36]である。「製品ライフサイクルの進行」に重点をシフトすれば，ソフトウェアが新奇的であるかどうかを捉える座標とし得る。その際，ソフトウェアのライフサイクル（開発～保守・運用～廃棄）と紛らわしいので，「ライフステージ」と呼ぶことにする。また，「顧客層」の区分・画期は，①革新派（Innovators），②早期適応派（Early Adopters），③前期多数派（Early Majority），④後期多数派（Late Majority），⑤無関心派（Laggards））であるが，ソフトウェアのライフステージの画期に相応しく改変する。①創生期，②普及初期，③普及拡大期，④コモディティ期，⑤レガシー期，とする。①創生期とは新奇的なプロダクトが誕生するステージであり，②普及初期とは先進的な利用が少数ながら開始されるステージであり，③普及拡大期とは利用が広範囲に広がるステージであり，④コモディティ期とは利用が飽和状態となり当たり前に使われるが差別化の余地は少なくなるステージであり，⑤レガシー期とは旧式化・陳腐化し衰退傾向にあり機会があれば別のプロダクトにリプレースされるステージである。

なお，ムーアを援用するのは，イノベーション研究において３ないし４段階に区分する一般的な「ライフサイクル」論に比べ，ソフトウェアの変遷・推移を捉えるのに最も適合的なものだからである。１つは，創生期と普及初期を区分することは，研究開発か否かを識別する上で決定的に重要である。２つには，普及初期と普及拡大期を区分することは，キャッチアップが先進的か否か，というソフトウェア業界の主要ターゲットを識別する上で有意義である。３つには，コモディティ期とレガシー期を区分することは，技術革新の激しいソフトウェアに関して，意外に思われるかもしれないが，単純に新旧交代がなされるのではなく，コモディティ期は比較的長く続き，またレガシーであっても長く延命する技術等もあり，それらを適切に識別するのに相応しいのである。

ソフトウェア・プロダクトの新奇性を捉えるためには，もう１つの分析的な枠組みが必要である，と考える。それは，ソフトウェアの機能と技術（内容と実現手段）という区分である。プロダクトとして出来上がったものは一体的であるが，分析的には区分可能であり，且つ区分しなければ新奇性が不分明になるおそれがあるからである。何故ならば，機能的には新奇的だが，技術的には広範に普及したもので些かも新奇的でないというプロダクトがあり，またその逆もあるからである。また，基本設計等において，ある規模以上の開発の場合，機能設計と方式設計はアクティビティとして分けられることも多く，実務的にも容易に区別することができるのである。会計基準における「技術的実現可能性」は，両者を識別せず，一体として取り扱っている。ソフトウェアを分析的に適切に捉えるには，両者を識別すべきことを強調しておきたい。なお，機能と技術が分解可能なことは，よく行われる異なったプラットフォーム（ハードウェア又はOS等）

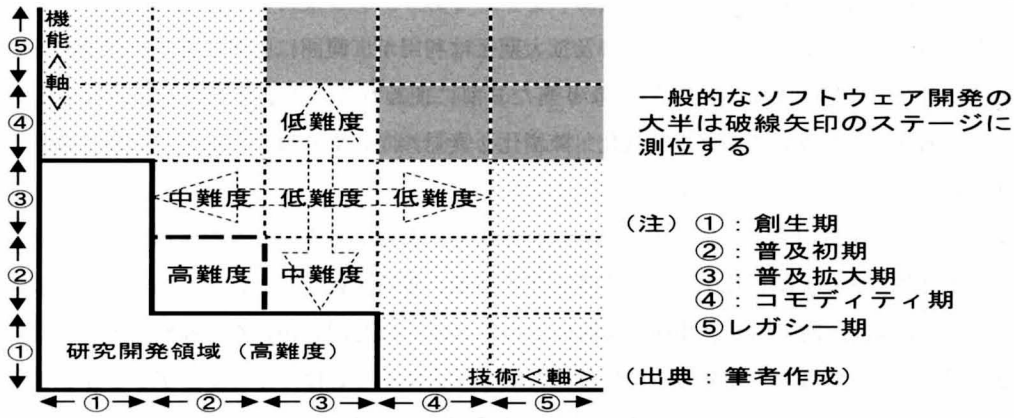
---

[36] Moore, Geoffrey（1991）邦訳 P.14

への移植（migration）を考えれば，理解しやすいであろう。同一の機能を異なった技術で実現することだからである。

これらにより，機能に関してライフステージを設定し，技術に関しても同様にライフステージを設定して，２次元の座標とすれば，開発対象であるソフトウェアの位相を特定することができる。それを図１に図示した。そして，機能並びに技術が総合的にみて新奇的か否か，従って当該ソフトウェア開発が研究開発であるか，それとも非研究開発であるかを，プロダクトの観点から明確に判断することができるのである[37]。

図１　ソフトウェアのライフステージ座標



２　ソフトウェア・プロダクトのライフステージ測位

ソフトウェア・プロダクトのライフステージに関して，注意を要することがある。１つは，全てのソフトウェア・プロダクトが，全てのライフステージを経過するわけではない，ということである。ある機能又は技術は，出現しても，大して普及もせずに消えていく（相当数はそうであろう），あるいは大した期間を経ずに急速に普及拡大し更にコモディティ化してしまうものもある（例えばクライアント／サーバ型システム等），あるいはレガシーと言われながらも，長らく

37　これに関してもう１つ参考としたアイディア源を紹介する。反復型開発方式の代表的な提唱者の一人であるウォーカー・ロイス（Royce,Walker）著『ソフトウェアプロジェクト管理—21世紀に向けた統一アプローチ（Object Technology Series）』（日本ラショナルソフトウェア監訳，ピアソン・エデュケーション刊行，1998年原著初版，1999年邦訳初版，2001年邦訳新装版）における「プロセスフレームワーク」である。ロイスは，ソフトウェア開発に如何にプロセスフレームワークを適合化させるかという観点から，種々の考察を行っている。その中で，「プロセスの多様性を表わす２つの主要な次元」として，「技術的複雑さ」と「管理的複雑さ」という２つの次元を設定し，それぞれの複雑さを高低で表わしている。ソフトウェア開発のプロセス間の違いを，プロジェクトの規模等の６つのパラメータで捉えている。ロイスの言う「技術」とは，機能と技術を包含した，広義の技術を意味する。ロイスの観点はプロセスであるが，プロダクトに重点を置き，分析的に捉える場合には，機能と狭義の技術に分解することが適切である。主要なアイディア源ではあるが，そのまま採用せず，独自の座標を設定することとした。

利用され続けているものもある（メインフレーム系等）。もう１つは，ソフトウェア・ライフサイクル（開発～保守・運用～廃棄）とは異なり，ライフステージはプロダクト自らが変化して変わるわけではなく，主としてソフトウェア（業界）の動向によって受動的にステージが変化することである。

さて，ライフステージの測位は，果たしてどの程度の客観性を有するか。１つに，『日経コンピュータ』等々の業界誌やインターネットの市場・技術動向情報等により確認できる。２つに，各業界における他社動向等は競争上相当に意識されキャッチアップがなされる傾向が強いので，各企業の採用状況から確認できる。３つに，開発実務に即した最も確かな裏付けとして，開発プロジェクトを編成する場合，採用する機能並びに技術の普及度によって，要員調達の難易度が大きく左右されるのである。従って，開発の難易度も左右される。このように，業界ないし市場，各企業，当事者（プロジェクト）の３つの視点から動向を捉えることによって，ライフステージの測位は十分に客観性を有すると言える。

測位に関して，表１に沿って説明を行う。第１に，機能又は技術が①創生期の場合が新奇性のあるプロダクトであり，その開発が研究開発である。片方が①創生期の場合，他方が④コモディティ期ないし⑤レガシー期にある場合を除外しているのは，販売可能性の観点からビジネスとしては考えにくいからである。機能又は技術が②普及初期の場合も同様である。機能が③普及拡大期にある場合に採用技術を⑤レガシー期とすることも同様である。第2に，機能が④コモディティ期ないし⑤レガシー期にある場合，（新規）開発は考えにくいが，利用されている限り，保守していくことはある。但し，④コモディティの機能を③普及拡大期の技術で開発すること並びにその逆は，低リスクなので多少はあり得るかもしれない。第３に，機能又は技術が②普及初期ないし③普及拡大期の場合が（新規）開発の圧倒的多数と考えてよいであろう。開発の難易度は前掲の図１に示した通りである。第１の場合以外は，何れも研究開発でないことは確実である。

以上によって，ステージの境界上に位置する等で判断が微妙なケースが若干あり得るとして

表１　機能と技術のライフステージの組み合わせ

（出典：筆者作成）

		技術				
		①創生期	②普及初期	③普及拡大期	④コモディティ期	⑤レガシー期
機能	①創生期	稀なケース	チャレンジ（リスキー）	チャレンジ（リスキー）	ビジネス的に考えにくい	ビジネス的に考えにくい
	②普及初期	チャレンジ（リスキー）	チャレンジ	○	ビジネス的に考えにくい	ビジネス的に考えにくい
	③普及拡大期	チャレンジ（リスキー）	○	○	安全第一で多少ありうる	ビジネス的に考えにくい
	④コモディティ期	ビジネス的に考えにくい	ビジネス的に考えにくい	安全第一で多少ありうる	開発はなく，保守の領域	開発はなく，保守の領域
	⑤レガシー期	ビジネス的に考えにくい	ビジネス的に考えにくい	開発はなく，保守の領域	開発はなく，保守の領域	開発はなく，保守の領域

も，個々のプロダクトを具体的に測位することは容易且つ高い蓋然性で可能である。次に取り上げるプロセスと総合的に捉えるならば，完全に判然とする。

Ⅳ　ソフトウェア・プロセスの様相

1　ソフトウェア開発のプロセス・イノベーション

研究開発は，新奇的な製品又は製法を追求する活動である。ソフトウェアに関しても，何ら変わらない。そこで，ソフトウェア・プロダクト（製品）に続いて，次にソフトウェア・プロセスの新奇性をどのように捉えるか，を明らかにする。それにより，個々のソフトウェア開発が研究開発であるか，非研究開発であるかを十全に明確に判断することができるからである。その際，2通りのアプローチが必要である。1つは，プロセス自体が新奇性を有しているか否か，ということである。もう1つは，プロセス自体に新奇性はないが，プロダクトが新奇的であるか否かで，それに適応的なプロセスが異なり，それによりプロセスとして研究開発か否かを区別する，ということである。

ここではまず，プロセス自体が新奇性を有する場合を概観する。ソフトウェア開発のプロセス・イノベーションを精緻に捉えるには，大別すると，①開発方式，②開発技法，③開発ツール，④プロジェクト管理という事項に分けて捉えることが適切である[38]。

①開発方式は，ウォーターフォール型，反復型（総称）としてインクリメンタル，プロトタイピング，スパイラル，アジャイル型等がある[39]。

②開発技法は，モジュール化，共通化，標準化，構造化，データ駆動開発，オブジェクト指向，モデル駆動開発，テスト駆動開発，アスペクト指向，コンポーネント開発，アーキテクチャ駆動開発，ドメイン駆動開発等がある[40]。

③開発ツールは，他の①②④と異なり，膨大な数がある。古くはコンパイラ（アセンブラを含む），テキスト・エディタ，デバッガに始まり，1970年代のCASEツールや今日のVisual Studioのような総合的なもの，更にはフレームワークと呼ばれる開発ツールでありつつ，開発するソフトウェアにコンポーネントとして組み込まれるものまで多数ある。それらを利用することがプロセスの範疇に属することであり，ツール自体はプロダクトの範疇のものである。そして，開発ツールは当該プロジェクトで自ら新奇性のあるツールを考案・作成し，且つ利用する場合を除き，既存のものを利用する限り，プロセスとしての新奇性はなく，研究開発には該当しない。

④プロジェクト管理は，PMBOK（Project Management Body of Knowledge：プロジェクト

38　この分類は，特定の文献等に依拠したものではなく，ソフトウェアに関する多くの文献並びに業界関係者から得た情報を元に，筆者が独自に抽出・整理したものである。

39　Pressman, Roger S.（2005）邦訳 P.35-55。但し，分類並びに呼称を簡略化のために，また筆者の分類である②開発技法を混在させているので，一部改変した。

40　開発技法の概説書は，筆者の探索の範囲では適当なものは見当たらなかった。個々の技法に関する解説書は夥しくあり，そのほんの一部を筆者は閲読したに過ぎないが，それらから抽出した。

マネジメント知識体系）を始めとして様々あるが，固有名が付いたものは余りない。

筆者の知る範囲で挙示したので，漏れはあるかもしれないが，特徴的なことは③開発ツールを除き，それ以外はソフトウェアの歴史60年ほど[41]を通しても，ごく限られた数のものが考案され，実用化されただけだということである。これらを考案し，実用化する取り組みは研究開発と言ってよいであろう。従って，プロセス（製法）自体が研究開発と言えるのは数える程のものである。それ以外は全て，プロセスに新奇性はなく，プロセスとしては研究開発の性格を有していないのである。

2　ソフトウェア開発と研究開発のプロセスの形態的相違

次に，もう1つのアプローチを行う。プロセス自体に新奇性はないが，プロダクトが新奇的であるか否かで，それに適応的なプロセスが異なり，それによってプロセスとして研究開発か否かを区別する，ということである。プロダクトのライフステージ測位と併せて，ソフトウェア開発が研究開発であるか否かを確実に捉えることができることになる。

端的に標語的に対比すれば，研究開発のプロセスは反復的（iterative）ないしループ的であるが，非研究開発は直線的（linear）である，ということである。その意味するところは，以下の通りであり，各々を図2，3に図示した。研究開発は，実績のない，成果が不確実なものを探求するのであるから，プロセスにおいて不可避的に試行錯誤を繰り返さざるを得ない。どの程度の試行錯誤の繰り返しを許容するかは，計画やマネジメントによるとしても，それを許容しない研究開発はあり得ない。研究開発マネジメントにより管理は強化される傾向にあり，短期開発並びにコスト削減のミッションも強まっているとは言え，試行錯誤の繰り返しは研究開発プロセスに構造的に組み込まれている。それに対して，非研究開発では，まず開発計画において試行錯誤の繰り返しが組み込まれることはない。レビューやテストというアクティビティやフェーズで誤りが摘出され，「戻り作業」が発生することはあるが，それによる進捗の遅延は許容されない。試行錯誤は実態的にはしばしば発生するが，あくまでネガティブに捉えられ，遅延の回復を強要される。最終的にプロジェクトの失敗（QCD（品質（Quality），コスト（Cost），納期又は開発期間（Delivery）の大幅な超過又は遅延・延長，プロジェクトの中止・中断）も少なくないが，研究開発のように成果が出ないことが計画に織り込み済みで許容されることはなく，成果が出ることが当然且つ通常のこととされている。

ここで起こり得る疑問を挙げ，それに応える形で事態を一層明確にしたい。1つは，一般的なソフトウェア開発を直線的と捉えるのに対して，それはウォーターフォール型[42]の開発のことで

41　コンピュータの創生は早くて1939年，ないし1940年代だが（Lattanze, Anthony J.（2009）邦訳P.27），ソフトウェアが独立的な意義を有するようになったのは1950年代に入ってからで，1つのメルクマールはプログラミング言語として，機械語より自然言語に多少近いアセンブリ言語が考案されたことであり，それから凡そ60年が経過している。

あって，反復型[43]もあり，必ずしもそうとは言えないのではないか，ということである。これは「反復」の意味・内容が異なるのであり，研究開発のような同一事項に関する試行錯誤の繰り返しを意味するわけではなく，異なる対象に対してアクティビティないしタスクを反復的に遂行するということである。簡単に模式的に図示すると，図３のようになる。反復型はアジャイル型を想定している。

もう１つは，研究開発のプロセスに関して，筆者のように反復的ないしループ的と捉えるのは一般的ではなく，「従来のモデル」は「第二次世界大戦以降，西欧で一般的に考えられているイノベーションの形態」の「リニアモデル」（linear model）であり[44]，それに対して「連鎖モデル」（chain-linked model）という「改良モデル」を提唱している，クライン（Stephen J. Kline）の有力な学説があるのではないか[45]，という疑問である。これは，同じプロセスと言っても，プロセス全体を俯瞰的に捉えたモデルに関する議論であり，筆者が問題にしているプロセスの内在的な作業レベルのこととは次元の異なる議論である[46]。同じ次元では，藤本隆宏が，研究開発における設計に関して，「２段階設計プロセス」モデルを提唱していることが参考になる。「公理系設計論をベースに，設計行為を，不確実性下で製品機能・製品構造の連立方程式を解く，２段階設計プロセスによって近似する。第１段階は，構造・機能の因果知識が不完全な中での暫定解の導出，第２段階は，その暫定設計解から，最適設計解へと漸近する試行錯誤のプロセスである」[47]（傍点：引用者）。独特の用語を使っているが，演繹的なアプローチにおいて，個別具体的な諸条件に適合させていく試行錯誤のプロセスを定式化したものであり，筆者の捉え方を支持するものと解して差し支えないであろう[48]。

---

42 ウォーターフォール型は，その名の通り，全般的に上流工程から順次下流工程に作業を進めていく開発方式であり，現在でも最も多く採用されている方式である。ポイントは，各工程で確実な成果物を作成し，後戻りしないことである。

43 反復型は，総称として括ったが，細かくは様々な流儀や名称があるが，大枠としては大同小異である。プロトタイプの作成から始めて，インクリメント（Increment：追加）並びにイテレーション（Iteration：反復）を繰り返して，段々に開発対象と範囲を拡大していく進め方である。個々のインクリメント並びにイテレーションのサイクルはウォーターフォール型と同様だが，大きな違いは，フル機能でなくとも，個々のサイクルの終了時にリリースまで可能なことである。

44 Kline, Stephen Jay（1990）邦訳 P.16。なお，通商産業省編（1992）も，「技術革新（ここでは，新製品の開発，新製法の導入等を指す）の過程は，従来，基礎研究から派生した新たな科学の知見が技術の芽となり，応用研究，開発を経て商業化へとつながる一本の直線的道筋（リニアモデル）でしばしば説明されてきた」（P.11-12），としている。

45 Kline, Stephen Jay（1990）邦訳 P.16。なお，研究開発マネジメントの世代論議に関しては原陽一郎（2009）P.7 等参照。

46 クラインの「連鎖モデル」でも，各フェーズ内並びにフェーズ間の「フィードバック・ループ」は設定されており，同一次元では捉え方は変わらない。

47 藤本隆宏（2011）P.275。

48 設計ループに関しては，Suh, Nam P.（1990）邦訳 P.24，39 参照。また，n 次試作に関しては岡本彬良（2005）P.19 参照。

図２　研究開発プロセスの模式図

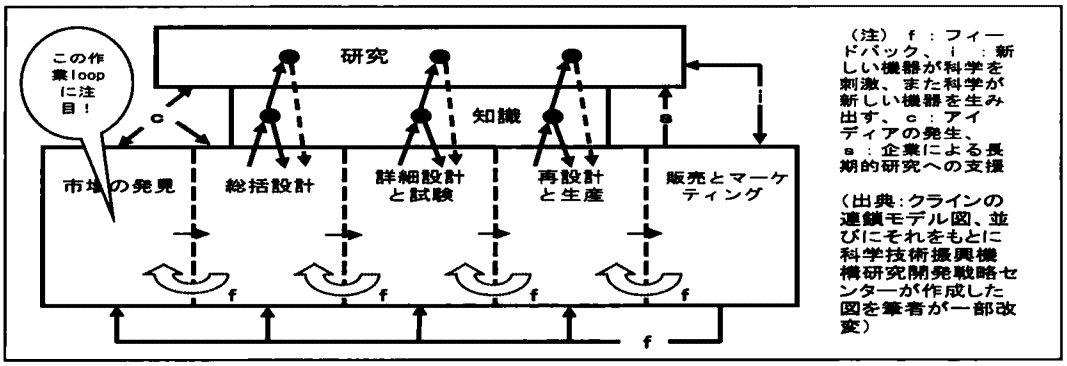
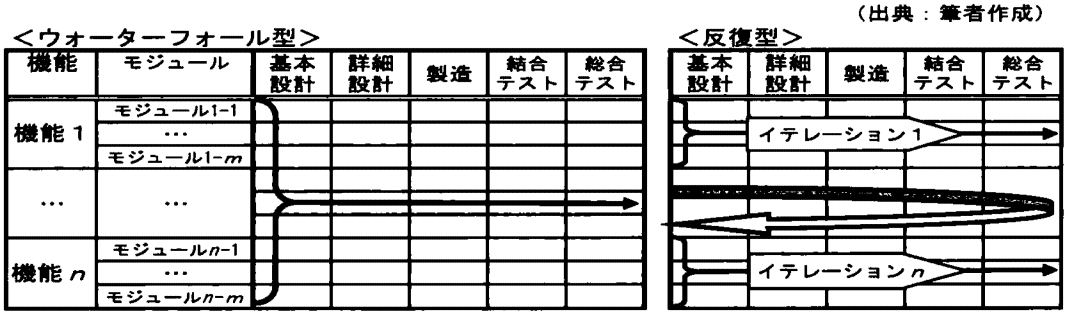


図３　一般的なソフトウェア開発プロセスの模式図

（出典：筆者作成）



３　ソフトウェア開発と研究開発のプロセスの定量的差異

ソフトウェア開発における研究開発と非研究開発のプロセスにおける形態的相違，すなわち試行錯誤の繰り返しの構造的な組込みの有無は，必然的に定量的な差異を生じさせる。それは，生産性の著しい差異である[49]。従って，当該開発の生産性が如何なる水準にあるかによって，研究開発であるか否かは判然とする。但し残念ながら，生産性に関する公表された信頼し得る統計的な情報はない。特に研究開発の生産性は企業機密に属することであり，具体的な数値を取り上げることはできない。そこで本稿では，モデル的な論証に留めざるを得ない。

簡単な例を挙げる。開発フェーズは，設計・製造（試作）・テストの３フェーズとし，工程比率は各30%・40%・30%とする。開発規模は120KLOCとする。

---

[49] 生産性（productivity）は，経済学の概念では，労働生産性と全要素生産性があるが（藤田昌久・長岡貞男（2011）P.4参照），一般的にソフトウェア開発で取扱う生産性（KLOC/人月又はFP/人月等）は労働生産性である。他方，研究開発で取扱う生産性は主としては全要素生産性である（事業成果（OP)/R&D投資（IP)，浦川卓也（2010）P.159参照）。但し，差異を比較考察する観点から，本稿では研究開発に関しても，ソフトウェア開発で取扱う労働生産性を個別プロジェクトに関して問題とする。

①一般的なソフトウェア開発の場合

・設計工数：45 人月，製造工数：60 人月，テスト工数：45 人月，合計 150 人月，とする。

・生産性は，0.8KLOC/人月（= 120KLOC／150 人月），となる。

・開発単価を 100 万円/人月とすると，直接開発費は 15,000 万円（=100 万円× 150 人月）である。

②研究開発の場合

・設計工数：1 回 45 人月を 3 回やり直すとすると，合計 135 人月，試作工数：1 回 60 人月を 3 回やり直すとすると，合計 180 人月，テスト工数：1 回 45 人月を 2 回やり直すとすると，合計 90 人月，となる。工数総合計は 405 人月となる[50]。

・生産性は，約 0.2963KLOC/人月（≒ 120KLOC／405 人月），となる。

・開発単価を 100 万円/人月とすると，直接開発費は 40,500 万円（=100 万円× 405 人月）である。

その差異が，歴然としていることがわかるであろう。例示では，研究開発の生産性は，一般的な開発対比 0.3704％と著しく低く，研究開発の直接開発費は，一般的な開発対比 2.7 倍と多額となる。公表された統計的数値はないが，例えば同一企業でソフトウェアを研究開発する場合と非研究開発する場合があれば，計画値並びに実績値は明確に差異があるはずである。

なお，一般的な開発でもプロジェクトの失敗によって大幅に低い生産性となる場合があり得るが，それは筆者が前稿[51]で提示した仕損会計の処理を施すべきことであり，本稿の考察に影響するものではない。また，もう 1 つ例外的な事態がある。研究開発で生産性がそれほど低くない場合である。天才的な，あるいは卓越した発想と能力を有する者が，常軌を逸した「全身全霊を賭した」とも言うべき献身により，作業に没頭して，ほとんど文字通り「不眠不休」で完成に辿り着くような場合であり，マイクロソフト，グーグル等の事例がある。しかし，そのような事例はあくまで「異例」のことであり，範例とはなり得ないと言うべきである。

V　おわりに

ソフトウェア会計基準において，およそ研究開発ではないソフトウェア開発まで不当に拡大して研究開発と看做す規定が，アメリカ基準を発端とし，日本基準でもそれを再検討することもなく踏襲された。ドグマティックに研究開発と看做すために，研究開発の本来の判断規準である製品又は製法の新奇性が軽視され，替わって「技術的実現可能性」の確保前後を工程で区分する判断規準が窮余の一策で策定されたが，それらが何れも根拠のない不合理なものであることを明ら

---

50 実務レベルでは，フェーズ単位というより，アクティビティないしタスク単位のより小さな作業単位でのやり直しの方が多いであろうが，モデル的に単純化した設定とする。計算の精粗や複雑さが異なるだけで，実質的なことは変わらない。

51 前掲「ソフトウェア開発の失敗に関する会計処理案」

かにした。日本基準制定前後に多くの論者が多くの論文等を発表したが，非研究開発を研究開発であると看做している規定に疑義を呈するものは僅かであった。その数少ない論考も，状況認識並びに発想としては概ね首肯できるものであったが，更に進展させて，具体的にソフトウェア開発が研究開発である場合とそうでない場合を明確に区別する判断規準を提示するまでには到らなかった。それは，今日でも変わっていない[52]。セントラル・ドグマは今も護持されている。

そこで筆者は，セントラル・ドグマを解体し，適正な基礎を構築するために，プロダクトとプロセスの両面から判断規準を明確化した。プロダクトに関しては，機能と技術のライフステージ座標を設定し，個々のソフトウェアを測位可能とした。プロセスに関しては，まず新奇的なプロセスの事例を挙示し，続いて大半のプロセスはそれ自体は新奇的ではないが，プロダクトの新奇性に適応的な研究開発とそれ以外の開発のプロセスが形態的並びに定量的に如何なる相違があるかを明確にした。これらにより，ほぼ完全に研究開発と非研究開発との区別を確実なものとした。

この考察の帰結は明解である。研究開発である場合は，研究開発費会計の対象として会計処理することでよい。それに対し，非研究開発の場合は，研究開発費会計とは別個の独立的なソフトウェア会計基準を制定し，その会計処理対象とすることである。そして，基準において資産要件を設定し，それを充足する場合には資産計上とする。なお，資産要件は過度に厳格である必要はない，と考える[53]。そもそも企業が効用もないソフトウェアの開発に少なくない投資をするなどと考えることが前提的に不自然である。また，一般的なソフトウェア開発にとって技術的実現可能性などは些少なことである。資産性の毀損に関しては，筆者が前稿で提示した「仕損会計」（当初）と，減損会計（それ以降）で対処すれば十分である。

本稿は，全面的に刷新されるべきソフトウェア会計の基礎的な考察であった。それに基づいた会計基準の体系的，包括的な試案を策定することは，今後の課題である。全面刷新されるべきソフトウェア会計基準は，ソフトウェアのライフサイクル（開発～保守・運用～廃棄）に沿った会計基準としなければならない。開発におけるソフトウェア再利用，開発ツールの利用，プロジェクトの失敗による仕損，保守，運用，廃棄を包含し，且つ複合的な実態を的確に捉えたものにしなければならない。また，市場販売目的と自社利用目的で大きく異なる規準ではなく，収益の源泉としての直間性に基づいた，統一的な規準としなければならない。陳腐化した，あるいは事実誤認の例示も刷新しなければならない。そもそも「日進月歩」は誇張だとしても，技術革新の激しいソフトウェアの会計基準が，制定から10数年経過しているにも関わらず，僅かな改訂しか成されていないこと自体が怠慢の謗りを免れない。例えば，オープンソースやクラウド・コン

---

52 企業会計基準委員会は，2010年時点では現行基準を「維持」しつつIAS38「無形資産」会計を重ね合わせた「二重基準」というコンバージェンスの方向を検討していたが（第197回企業会計基準委員会（2010年3月11日）審議事項（5）－1（https://www.asb.or.jp/asb/asb_j/minutes/20100311/20100311_index.jsp)），2012年時点では現行基準を「維持」し，変更しない方向を鮮明にした。

53 IAS38の資産要件は一見包括的で周到なようで，研究「開発」と重ね合わせているので，ソフトウェア開発に関しても過度な要件となっている。

ピューティングが会計的に惹起している問題に対処する必要性を，筆者は痛感し研究を進めている。それらを完成した暁には全面刷新案を提案したいと考えており，本稿の内容はその核心部分を構成することになることは言うまでもない。

最後に，繰り返しになるが，ソフトウェア開発を研究開発と看做すセントラル・ドグマから解放され，ソフトウェア会計が正常化されるために，本稿が一石を投じることを念願している。

引用文献

・浦川卓也（2010）『実践研究開発マネジメント』日刊工業新聞社
・岡本彬良（2005）『よくわかるプリント基板回路のできるまで―基板設計，解析，CADからDFMまで―』日刊工業新聞社
・企業会計審議会「研究開発費等に係る会計基準」1998年
・企業会計審議会「研究開発費等に係る会計基準の設定に関する意見書」1998年
・櫻井通晴編（1993）『ソフトウェア会計　ソフトウェア会計実務指針「案」の解説と実際例』中央経済社
・櫻井通晴（1999）「ソフトウェア会計の基準化は何をもたらすのか――日本企業に及ぼすインパクト」『経理情報』1999年7月1日号
・櫻井通晴（2012）「IFRSがソフトウェア開発費の会計処理に及ぼす影響――ソフトウェア開発費の理論的・実務的検討」『企業会計』Vol.64 No.8
・通商産業省編（1992）『産業科学技術の動向と課題　地球規模での技術的共生に向けて』通商産業調査会
・天明茂（1996）「汎用ソフトウェアの会計処理に関する一考察――製品マスターの資産性と棚卸資産表示の妥当性吟味」『企業会計』Vol.48 No.5
・中村恒彦（2003）「アメリカにおけるソフトウェア会計の経路依存」『桃山学院大学経済経営論集』第45巻第2号　桃山学院大学総合研究所
・西澤脩（1991）「米国におけるソフトウェア費の会計処理基準」『早稲田商学』通号349
・日本公認会計士協会「会計制度委員会報告第12号　研究開発費及びソフトウェアの会計処理に関する実務指針」1999年
・原陽一郎（2009）「第3世代の技術経営（MOT）…組織を超えたマネジメント・システム」『長岡大学　研究論叢』第7号　長岡大学
・藤本隆宏（2011）「設計比較優位説のプロセス的基礎」（藤田昌久，長岡貞男編著『生産性とイノベーションシステム』日本評論社
・FASB, *Statement of Financial Accounting Standards No.86 "Accounting for the Costs of Computer Software to Be Sold, Leased, or Otherwise Marketed"*, 1985.（財務会計基準審議会（日本公認会計士協会　国際委員会訳）「財務会計基準書第86号　販売，リースその他の方法で市場に出されたコンピュータ・ソフトウエア原価の会計処理」1985年）
・Jones, Capers (1995), *Patterns of Software Systems Failure and Success*, International Thomson Computer Press, Boston.（ケーパーズ・ジョーンズ（伊土誠一・富野壽監訳）(1999)『ソフトウェアの成功と失敗』構造計画研究所，共立出版）
・Kline, Stephen Jay (1990), I*nnovation Styles In Japan and The United States cultural bases; implications for competitiveness*, Stanford University（S. J. クライン（鴨原文七訳）『イノベーション・スタイル　日米の社会技術システム変革の相違』　アグネ承風社　1992年）
・Lattanze, Anthony J. (2009), *Architecting software intensive systems: a practitioners guide*, Taylor&Francis Group,LLC, Boca Raton.（アンソニー・J・ラタンゼ（橘高陸夫監訳）2011『アーキテクチャ中心設計手法　ソフトウェア開発の実践　ソフトウェア主体システム開発のアーキテクチャデザインプロセス』翔泳社）
・Lev, Baruch (2001), *INTANGIBLES; Management, Measurement, and Reporting*, Brookings Inst Pr, Washington,D.C.（バルーク・レブ（広瀬義州，桜井久勝監訳）2002『ブランドの経営と会計』東洋経済新報社）

・Moore, Geoffrey. A. (1991), *Crossing The Chasm: Marketing and Selling Technology Products to Mainstream Customers*, Harpercollins, New York（ジェフリー・ムーア（川又政治訳）『キャズム　ハイテクをブレイクさせる超マーケティング理論』翔泳社　2002年）
・Pressman, Roger S. (2005), *Software Engineering: a practitiners approach*, 6th edition By McGraw-Hill, New York（ロジャー・S・プレスマン（西康晴，榊原彰，内藤裕史監訳，古沢聡子，正木めぐみ，関口梢訳）『実践ソフトウェアエンジニアリング　ソフトウェアプロフェッショナルのための基本知識』日科技連　2005年
・Suh, Nam P. (1990), *The Principles of Design*, Oxford University Press, New York（N・P・スー（畑村洋太郎監訳）『設計の原理　—創造的機械設計論—』朝倉書店　1992年）